Conference Proceedings

On 17–18 April 1962 the Northampton College of Advanced Technology, London, in co-operation with The British Computer Society, held a conference on

Automatic programming languages for business and science

Complete Proceedings of Sessions 1 and 3 of this conference are published on the following 33 pages of this issue. It is hoped to publish the Proceedings of Sessions 2 and 4 in the October issue.

Current developments in commercial automatic programming

By A. d'Agapeyeff

This paper discusses the progress made in certain aspects of commercial automatic programming, presents a progress report on the major commercial languages, and offers some hopes and expectations for the future.

Introduction

It is of course quite impossible, in the time available, to give a complete survey of current developments even supposing one had the ability to do so. I intend, therefore, to confine myself to three headings, namely:

(i) progress in certain aspects of commercial automatic programming;
(ii) progress report on the major commercial languages;
(iii) a look at the future.

Before continuing it is perhaps necessary to give a warning that this paper does not consist of a repetition of the silver-lined platitudes which tend to occur on these occasions. It will instead attempt to provoke a little realism, of which there appears to be some need. At the same time it is not to be taken as in any way a denial of the importance of automatic programming, or of the great urgency that exists in attaining its objectives.

Progress in Certain Aspects of Commercial Automatic Programming

It is seldom that anyone attempts to consider what are the different aspects of a commercial automatic programming system. Most interest is usually concentrated on the language as such (e.g. the grammatical rules or the particular facilities that are currently in fashion). Thus arguments rage as to whether one should say

$$c := a + b, \quad \text{or}$$

ADD a TO b GIVING c

although this is not of the slightest real importance to the compiler, providing that the intended meaning is known precisely.

Yet a programming language obviously amounts to more than a few tasteful pieces of notation. We can perhaps best appreciate this by considering the four questions that ought to be, but seldom are, asked of any such system:

(i) What are the properties that units of data may be given and how, if at all, are these to be declared?
(ii) What are the procedural facilities available? How are these denoted and how can they be combined into program statements?
(iii) How is communication obtained with the object program, after it has been compiled, and how is it to be debugged?
(iv) Is the compiler available to the user on his own machine and, if so, how is it operated?

The properties of data

It is, of course, the available properties of the data which to a large extent determine the power of an automatic programming system, and distinguish commercial from mathematical languages.

Consider the function of moving data within the internal store. In a mathematical language the problem is trivial because the unit which may be moved is very restricted, often to the contents of a single machine word. But in a commercial language this limitation is not acceptable. There, data units will occur in a variety of shapes and sizes, for example:

(i) Fixed Length Units (i.e. those which on each occurrence will always be of the same length) may vary widely in size and will tend not to fit comfortably into a given number of words or other physical unit of the machine. Generally the move will be performed by a simple loop, but there are some awkward points such as what to fill in the destination if the source is the smaller in size;

(ii) Static Variable Length Units (i.e. those whose length may vary when they are individually created but will not change subsequently) are more difficult to handle. Essentially the loop will have two controlling variables whose value will be determined at the moment of execution. There are again awkward points such as the detection of overflow in the destination (and deciding what to do when it occurs, since this will only be discovered at run time);

(iii) Dynamically Variable Length Units (i.e. those which expand and contract to fit the data placed in them) are even more difficult. They have all the problems of (ii), together with the need to find and allot space when they expand.

It is clear, therefore, that a simple MOVE is less innocuous than it might seem at first. Actually the above remarks assumed that it was not possible to move data between different classes of units. The absence of this restriction, and the performance of editing functions during the process, can make the whole thing very complicated for the compiler indeed.

The properties of data will have a similar influence on most of the other operators or verbs in the language. This has particular significance when the desired attribute is contrary to that pertaining on the actual machine. Thus arithmetic on decimal numbers having a fractional part is thoroughly unpleasant on fixed-word binary machines.

Nevertheless, despite these difficulties considerable progress has been made toward giving the user the kind of data properties he requires. Unfortunately this progress has not been matched by an improvement in machine design so that few, if any, of the languages have achieved all of the following.

(*a*) The arbitrary grouping of different classes of unit, allowing their occurrence to be optional or for the repetition in variable-length lists.

(*b*) The input and output of arbitrary types of records, or other conglomerations of data units, having flexible formats, editing conventions and representations.

(*c*) The manipulation of individual characters, with the number and position of the characters being determined at run time.

(*d*) The dynamic renaming or grouping of data units.

Yet users do need these facilities. It is not always acknowledged that getting the main files on to the computer, before any processing is done, may constitute the largest single operation in many applications. Furthermore, these files will have a separate independent existence apart from the several programs which refer to them.

Progress has also been made in declaring the data properties in such a way as to imply a number of necessary procedures to the compiler. For example, if one declares both the layout of some record to be printed, and the printed record, the compiler may deduce the necessary conversion and editing processes. It is here, in the area of input and output, that some languages have approached the aim of being problem-orientated.

Procedures

The procedure division constitutes the language form, and is dominated by the rules of grammar and the facilities available. This is the area which has been the cause for the fable that anyone can now program computers, but it also provides the grounds for refuting this claim.

At first sight it does appear easy to write down simple English language statements, and for the management to follow what has been written. (A friend of mine was complaining bitterly the other day that his life was a lot easier when he programmed in machine code, and the management could not pretend they knew what was going on.) The unpleasant truth is, of course, that none of the major languages are as easy to learn as a really simple machine code, such as the Elliott 803. They are all full of rules and exceptions, although once these have been learnt, programs may be much easier to write and above all to document. That is in fact the point: one is still programming, albeit in a more natural way, and skill is still necessary if anything worth while is to be achieved.

The worst feature of most procedure divisions is the lack of adequate facilities to define new functions. These are necessary to the user because, sooner or later, he is bound to come across a situation which the compiler-writers had not envisaged. In these circumstances he may well find that either he literally cannot do what he wants, or he must do it in a most cumbersome or unnatural way. Here ALGOL is greatly in advance of any existing commercial language.

Communication with the Object Program

This aspect has received scant attention from either manufacturers or the users. All too often there is a

blithe assumption that one compiles a program and that is the end of the matter.

Consider two obvious requirements.

(i) It should be possible to write routines referring to data which will be identified at run time. For example, one would like to produce a routine to handle inquiries to any data unit within a given record, without having to print out the entire record each time. At present this cannot be done by a single generalized routine because the data description is not available during the program run (it would not, however, be impossible to print out a coded data description which the user would put in with his inquiry).

(ii) It also should be possible to choose what routines are required in a given program run so that only those are loaded on to the machine.

In regard to debugging, the only really acceptable form of tracing the execution of a program is that of the language in which the program was originally written. This is a very difficult problem which has not yet been solved in practice, but I am convinced it is both possible and feasible, although expensive, in terms of compiler effort.

Using the Compiler

Most of the current language systems allow the user to obtain a copy of the compiler, but there are often restrictions as to the size of machine on which it will run. The actual process of compilation is generally somewhat lengthy, and after it has been completed (assuming no source errors have been found which would probably necessitate starting all over again) voluminous paper is produced reporting on the object program. Naturally none of this information means anything to the user unless he knows at least the assembly code, if not a good deal about the machine and the methods of compilation.

It is at this point that the full flavour of the "Alice in Wonderland" situation we are now in becomes apparent. A great deal of time and money is spent to support the claim that "any fool can program this computer." The realization that it takes a special kind of fool to compile or debug it is left until Alice awakes, by which time she has presumably paid the bill!*

Progress Report on the Major Commercial Languages

COBOL

One must, of course, start with COBOL if for no other reason than because sooner or later, for good or ill, we are all going to be concerned with it.

In view of all the publicity and argument I presume that everyone has heard of COBOL: how it was started by the American Defense Department in conjunction with the majority of American manufacturers; how pressure was put on its acceptance by the Defense Department (the largest single user of computers in the world) through their insistence that they would not buy any computer which did not have a compiler for it.

It was natural for the American manufacturers to press for the universal adoption of COBOL due to the large expenditure they were committed to by its implementation. It is true that some were a little cool towards it at first, but all now seem reconciled.

The opposition to COBOL depended on the attitude of I.B.M. much in the same way as does the opposition to ALGOL. I.B.M. have announced that they will not implement their own system called COMMERCIAL TRANSLATOR for any new machine. If this means they are going over entirely to COBOL then that will in my opinion be decisive. But up to now the acceptance of the language among American user groups, such as the SHARE organization, has not been as widespread as the originators would have wished.

As for COBOL itself, the language has made progress. The 1961 version is much tidier and better described than in 1960. A more realistic attitude has been taken to the degree of commonality across machines that can be obtained, and a definite attempt has been made to get everyone to implement the same language and so end the profusion of COBOL-like dialects that followed the 1960 publication.

Yet a number of problems remain. The language is still not specified with a sufficient degree of precision. It lacks a means of defining new functions of adequate generality. Further, although the next report is not due until 1963, there is a very real difficulty in preventing new versions, in so far as they are a change rather than an extension to previous ones, from causing the kind of confusion the language was designed to prevent.

Nevertheless COBOL, unlike ALGOL, is a complete programming system and in this, and the ideas it has introduced, it does represent a considerable achievement. Many of those in this country (myself included) who were engaged in designing a different language learnt a great deal from it. It is already available here, mostly in forms only approximately to the 1960 report, but the 1961 compilers will soon be available from across the Atlantic.

FACT

This is the language of Minneapolis–Honeywell and is currently working on their 800, newly installed at Moor House.* It is an untidy language, poorly supported by both its general and reference manuals.

But the looks of FACT belie its power. For FACT is a real programming tool which is aimed at the important practical problems of the user. It has tackled file processing in a big way, although the rules may be confusing at first, and it has probably the most powerful facilities for input and output of any existing language. Like COBOL it lacks, however, a suitable means of defining new functions.

* It takes a wise man to be a fool at court—*Ed.*

* More complete details of FACT will be found in the paper by Dr. R. F. Clippinger beginning on p. 112 of this issue.—Ed.

In comparing FACT to languages developed in the U.K. it must be remembered that it involved an effort of at least an order of magnitude greater than was ever available on a home project of this kind. There are some 220,000 three-address instructions in the compiler, and the mind boggles at how such a program was ever organized and debugged. I believe this has been a problem, and that work still continues on completing final points.

COMMERCIAL TRANSLATOR

This is a language of I.B.M.; it is currently working on the 705 and 7070 systems and is committed for the 7090. It is a simple but elegant language supported by an excellent manual. It contains both adequate means for defining new functions and other facilities not yet available in COBOL, but rather surprisingly it does not allow variable-length fields.

COMMERCIAL TRANSLATOR does not have anything approaching the facilities of FACT for input and output. Presumably they are available on the systems tape or otherwise as software packages, but it is in consequence impossible to make a proper comparison between the two languages. Actually they are both very good in different ways, and if COBOL-63 combines the best of both it will indeed be a step forward.

Languages developed in the U.K.

The remaining languages (except the last) are all home products, and there will in consequence be no surprise to find that they are either not working, or not on a par with their American equivalents. Prospective users of these languages should remember that, on past experience, the compilers cannot be expected to be fully debugged until at least six months after they are first run—assuming that happy event ever occurs, which is by no means certain in all cases due to the prevailing difficulties of attaching magnetic tape to current British machines.

RAPIDWRITE

This is the language of I.C.T., developed initially for the 1301 but now to be also implemented on the 1500, although this machine has, of course, a COBOL compiler supplied by R.C.A.

RAPIDWRITE is a subset of COBOL based chiefly on the 1960 version but with some facilities apparently introduced in anticipation of the 1961 report. Its chief merit is the reduction of writing and punching obtained by the use of pull cards for program statements, and yet the compiler produces a full COBOL listing for documentation purposes.

RAPIDWRITE in its minimum form (i.e. for the smallest 1301 without magnetic tape) is likely to be the first of the advanced home-made commercial compilers that actually works. And, what is more, the compiler will itself operate on the minimum machine.

NEBULA

This is the language of Ferranti designed originally for the ORION but now to be also implemented on ATLAS. NEBULA has some very nice facilities, particularly in regard to input and output and the general use of formulae. Unfortunately, like FACT, it has grown a little untidy while the compiler has progressed, and there are now a great many rules in the language. It may well, however, prove to be a valuable tool once it is working.

CLEO

This is the language of LEO Computers, designed for LEO III, and is almost certainly the most recent language to be specified. It is of particular interest due to its having a combined aim toward both mathematical and business purposes.

Only a preliminary report on CLEO has so far been available, but this is impressive in so far as it goes. The language might be described as being akin to both FACT and ALGOL, in that there are features of the former in the file processing and of the latter in the other procedures.

FILECODE, and LANGUAGE H

We now come to the two English projects which are actually working. FILECODE is the language implemented on Sirius and promised for Pegasus; while LANGUAGE H is working on the National-Elliott 405 and intended for the 315 and 803. Both are a little primitive and their facilities cannot be compared to those contained in the languages described above. FILECODE would appear to be the major effort because it incorporates a Data Division; without it LANGUAGE H might be regarded as an elegant English-language assembly code.

However, the simple fact that both languages work is in itself grounds for congratulating their originators. Both must be providing useful practical experience; and one further admires the realistic way these projects were tackled with the minimum of advanced promises.

ALGOL

Finally, in this section some mention might be made of the work done towards extending ALGOL for commercial purposes. Nobody now seems very interested in this effort, but for myself, and I had some hand in it, I still believe it could be valid and hope it will soon be published.

ALGOL is a really elegant tool for programming, or would be if those responsible for it would stop squabbling long enough to complete it. There is no reason why the language should not be extended to provide a unified system for both business and mathematical purposes. The difficulty of notation for business users could be removed quite trivially, by including an alternative "English" word form. Furthermore, ALGOL is the

only existing language in which the processes of compilation could be described with the precision that is necessary if we are ever going to achieve a real compatibility across different machines.

A Look at the Future

It is perhaps a natural optimism that prevents one looking at the future without some kind of hope. I would therefore offer the following.

Hopes

(i) That business computers will cease to be rehashes of the old arithmetic machines of von Neumann, Wilkes and others, and come to be designed for the tasks that face them. The Burroughs B5000 may show the trend by restricting users to the source languages provided. Then presumably one can stop the salesmen being interested in any way with the form of the machine instruction code, and make it suitable for the compiler-writers.

(ii) That certain manufacturers will lessen their attempts to provide solutions to commercial problems based on languages which are incompletely specified, compilers which are not written, and computers that do not exist; and that some other manufacturers will discover there is a need for commercial automatic programming.

(iii) That the profusion of committees working [*sic*] towards the standardization of programming languages, under such bodies as E.C.M.A. (European Computer Manufacturers Association), I.F.I.P. (International Federation for Information Processing), and I.S.O. (International Standards Organisation) will be combined into a single effort. Such a combination to begin, as belatedly a B.C.S. group is now engaged, by deciding what exactly the standardization of programming languages means.

(iv) That someone will write an interpreter for COBOL of about 5,000 instructions, taking a few man-months of effort, and then find the object programs are so much shorter that, in certain circumstances, the result is actually more efficient than present methods.

(v) That British users will come to realize that the acquisition of these new and expensive forms of office plant carry with them the responsibility of learning something about them, and in particular about programming *per se*; that the larger users such as the banks, insurance companies, and the nation-wide distributors will appreciate the need to build their own programming systems, since the manufacturers cannot be expected to cater for all their individual requirements.

Expectation

So much for hope; expectation is another matter.

(i) I expect a standardization of languages, whatever that may mean, to occur within the next year or two, because that is what the salesmen want, and they, after all, are the people who decide these matters.

(ii) I expect COBOL to be accepted as *the* commercial standard rather than *a* standard, due to a general lack of appreciation of the importance of the distinction between these two alternatives.

(iii) I expect this country to become more and more dependent not only on American machines but also on American programming systems. The programming efforts on the two sides of the Atlantic are so disproportionate as to make this result inevitable. And perhaps that is what we deserve, because certainly we are not now doing sufficient to justify a separate existence, or even possibly to call ourselves a home industry.

Bibliography

WILLEY, E. L., *et al.* (1961). *Some Commercial Autocodes—A Comparative Study*, APIC Studies in Data Processing No. 1 Academic Press.

D'AGAPEYEFF, A., *et al.* (1961). "A Critical Appraisal of COBOL," *Annual Review in Automatic Programming*, Vol. II.

D'AGAPEYEFF, A., *et al.* (1962). "Progress in Some Commercial Source Languages," *Annual Review in Automatic Programming*, Vol. III.

COBOL-61. Report published by Department of Defense, Washington D.C.

FACT *Manual* (Interim Edition). Minneapolis-Honeywell DSI-27E (1961).

I.B.M. COMMERCIAL TRANSLATOR. General Information Manual F28-8043 (1960).

I.C.T. RAPIDWRITE (Programming Manual). International Computers and Tabulators Limited. P.155/9.61/5M/SL, 1961.

HUMBY, E. (1962). "Rapidwrite—a New Approach to COBOL Readability," *The Computer Journal*, Vol. 4, p. 301.

NEBULA—a programming language for Commercial Data Processing, Ferranti LD12 (November 1960).

BRAUNHOLTZ, T. G. H., *et al.* (1961). "NEBULA: a Programming Language for Data Processing," *The Computer Journal*, Vol. 4, p. 197.

CLEO—Leo Computers Limited, November 1961.

"FILECODE"—Ferranti Limited. C.5325.

EXPERIMENTAL DATA PROCESSING LANGUAGE, National Cash Register Co. (1960).

BACKUS, J. W., *et al.* (1961). "Report on the Algorithmic Language ALGOL 60," *Annual Review in Automatic Programming*, Vol. II.

DIJKSTRA, E. W. (May 1961). "Making a Translator for ALGOL 60," *Automatic Programming Information*, No. 7.